About objects and approaches
a conceptual view on building models

Ann Hendricx and Herman Neuckermans
CADLAB, Department of Architecture, Urban Design and Planning, KULeuven, Belgium

Key words: CAAD, integrated design environment, building models, conceptual modelling, MERODE

Abstract: Considering integrated CAAD environments for architectural design, a number of different approaches are possible. This paper presents the policy of the CADLAB at the KULeuven University, where design support right from the first design phases is a basic consideration. After a short introduction on the theoretical framework and additional design tests, we will discuss the core object model that forms the cornerstone for the contemplated design environment. This object model describes all possible data, concepts and operations connected with the architectural design process. For its development, we used the object-oriented analysis method MERODE. The starting-points and main aspects of the model will be discussed, illustrated with examples of implemented prototypes. The architect's point of view and the specific nature of the architectural design process were always kept in mind, thus leading to a model that hopes to make a valuable contribution to the research area of integrated design environments.

1. INTRODUCTION

1.1 The computer in the design process

Architects use a great variety of concepts while designing: geometrical shapes, spaces, building elements, user activities, structure, etc. Within this vocabulary, different design approaches are possible. Some designers follow a top-down procedure; others proceed according to a bottom-up strategy. Whatever the
approach, it goes without saying that commitments made in the first stage of the design process have the biggest impact and are the hardest to undo later.

Until now software packages offer a broad range of computational tests to evaluate an architect’s creation in the ‘post-design’ stage: adequate tests capable of handling incomplete data and estimates are rare. Exactly those tests could help the design process in the first stages. Additionally, we should not ignore the fact that in today’s practice the computer only comes into play when the design is more or less completed. Computation programmes and the drawing packages themselves should concentrate more on this first stage of sketching and exploration. At least the unproductive translation of a design made by hand into a digital version would be avoided.

To assist the architect throughout the whole design process, an integrated approach is essential. To modelling tools and testing tools alike, they should be part of one and the same design environment, providing the architect with full interactive use of the tools while working on one and the same design model. Both in the field of product modelling - the description of the designed product - and in the area of design and testing tools (e.g. Sequin et al. 1998) a lot of efforts are carried out world-wide. But often the applicability of these efforts is limited: schematic and poor design possibilities, forcing the designer into a specific way of working, concentrating on a particular evaluation test and thus hampering later extension of the system etc. Approaches often underestimate design as a process, and neglect the evolutionary nature of design.

An integrated design environment basically consists of the following parts (Kim et al. 1998, Eastman et al. 1997):

1. a core object model or aggregation of models able to describe all actors and processes within the world of architectural design
2. a data management system
3. tools to assist the architect while designing

We like to subjoin the following remarks: First, we enlarge the chance of success by leaning on an underlying theoretical framework. In this way, the integrated approach is secured and developing a complete environment will be easier. Our proposal for such a framework will be discussed further down.

Next, we divide the assisting tools in those tools enriching and helping the design process itself and those tools evaluating and appraising designs. The first category includes both modellers - a fine example is the very attractive Sculptor, a still evolving package that encourages an intuitive way of handling bodies in space (Kurman 1998) - and research topics such as shape grammars, case based reasoning, automatic layout configurations etc. The second category covers tools for predicting thermal behaviour, lighting, insolation, statics etc.

The IT approach is not only beneficial to the designer, all building partners will take advantage of the integrated environment by referring to or using the core product model. A big step forward knowing that communication and passing on of data between professionals in the building industry often happens in a time-consuming and inefficient way. In this respect, research on data transfers and collaboration - also within the design team - and our research join hands.
1.2 A theoretical framework for the architectural design process

Considering these preliminaries, a general conceptual model for computer-aided architectural design has been developed (Neuckermans 1992).

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Masterplan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Design Entities</td>
</tr>
<tr>
<td>Geometry and attributes</td>
<td>Topology and attributes</td>
</tr>
<tr>
<td>Basic building scope</td>
<td>Cost/m² or cost/m²</td>
</tr>
<tr>
<td>Masterplan scope</td>
<td>Surface and volume/space</td>
</tr>
<tr>
<td>Site contingencies</td>
<td>Temperature fluctuations</td>
</tr>
<tr>
<td></td>
<td>Level of insulation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 2</th>
<th>Block or Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Design Entities</td>
</tr>
<tr>
<td>Geometry and attributes</td>
<td>Topology and attributes</td>
</tr>
<tr>
<td>Rooms or singular spaces</td>
<td>Cost/m² based on takes</td>
</tr>
<tr>
<td></td>
<td>Surface and volume/space</td>
</tr>
<tr>
<td></td>
<td>Temperature fluctuations</td>
</tr>
<tr>
<td></td>
<td>Morphology</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 3</th>
<th>Room or Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Design Entities</td>
</tr>
<tr>
<td>Geometry and attributes</td>
<td>Topology and attributes</td>
</tr>
<tr>
<td>Building elements:</td>
<td>Cost/m² based on takes</td>
</tr>
<tr>
<td>Wall</td>
<td>Surface and volume/space</td>
</tr>
<tr>
<td>Column</td>
<td>Temperature fluctuations</td>
</tr>
<tr>
<td>Beam</td>
<td>Morphology</td>
</tr>
<tr>
<td>Arch</td>
<td>Comfort prediction</td>
</tr>
<tr>
<td>Door</td>
<td>Daylighting</td>
</tr>
<tr>
<td>Stair</td>
<td>Sunshining</td>
</tr>
<tr>
<td></td>
<td>Elementary stability</td>
</tr>
</tbody>
</table>

Figure 1. A conceptual framework for the design process

In this model, the building programme is subdivided hierarchically into 3 levels spanning the normal scope of architectural design. Extension of these levels both upward to the urban design scale and downward to the building element is possible, when judged relevant by the designer. Depending on personal preferences or on the design problem at hand, each level can be seen as an entry point for the design process. On any level, the architect can use entities he is familiar with. Building types, spaces and elements like walls, columns, doors and stairs take the place of polylines, ruled surfaces and other "meaningless" CAD objects. Optionally, these elements can be placed on a grid, whose mesh widths vary with the actual level of detail one is working on. In addition to levels, elements and grids, the framework provides a collection of tests allowing the user to analyse building properties. These tests are in tune with the precision of the model at that moment and are relevant to the specificity of the building programme (hospital, school, office, housing etc.). This model presented the notion of a scale dependent representation: the 'intelligent zoom' displays one and the same object with more detail when the scale of the drawing increases (Neuckermans, 1986).
Globally, this framework remains primarily graphical and is an open, not deterministic but interactive aid to the designer. It aims at assisting instead of steering the design process.

2. **THE CORE OBJECT MODEL**

Based on the logic of this theoretical framework, the first research phase led to the development of a conceptual object model able to describe all elements mentioned in the framework. After a short introduction to MERODE, the analytical method used to develop the model, we discuss the most important aspects of the model. This part is illustrated with samples of code and images of implemented prototypes.

2.1 **MERODE, an object oriented analysis method**

We have chosen to use MERODE, an object oriented analysis method to produce so-called business models because of its elaborated systematic approach. This approach has proven to be a great help to achieve the contemplated core model. We may as well mention that we were charmed by the fact that the authors specifically refer to architectural design when describing the development of business models (Zachman 1987, Dedene et al. 1996).

MERODE is an acronym for Model driven Existence dependence Relation Object-oriented DEvelopment. For a full description of the method we refer to Verhelst (1996). Since some aspects of our object model are a direct consequence of this method, we tersely mention its main features.

First, MERODE makes a clear distinction between the specification phase and the implementation phase. The specification phase is entirely object-oriented. It makes no assumptions whatsoever about technology that will be used for implementing the system. This means that specification is completely conceptual and therefore independent of technological aspects of a system. Both ‘traditional’ implementations and object-oriented ones are still possible. Thus, tough choices about relational or object-oriented databases, CAD packages and so on are avoided at this stage of the design. And more important, research is not limited by the chosen hardware and software or left to the arbitrariness of a certain software developer.

During the conceptual specification phase, reliability, ease of correcting and flexibility are the relevant quality measures, whereas during implementation performance, efficiency and user-friendliness are prevailing. In specifying MERODE models you even need to have no knowledge whatsoever about computer-related matters. After specification, a computer specialist can carry out implementation with minimal effort. This supports the idea that implementation is achieved by transforming the specification and not by further elaboration.

Second, a distinction is made between ‘business objects’ and ‘function objects’. Business objects correspond to objects that exist in the real world. Defining methods attached to the business objects enforces business rules. Thus, the business subsystem simulates the real world system (e.g. a car rental company). When real world rules modify, only the business subsystem has to be changed. It can never be
influenced by changes in information needs (subject matter of the output subsystem) or by changes in business organisation (subject matter of the input subsystem). In the example of the car rental company, ‘car’ and ‘customer’ are examples of business objects. Apart from these business objects, function objects can be of three kinds:

- **Input objects** gather the necessary information related to the actual system and send messages to the business objects.
- **Output objects** gather information from the business objects and present it to the user in an appropriate way, e.g. printing an invoice.
- **Information objects** hold track of an object’s history. Because the business model only simulates the real-time situation, this information is necessary for example when you need to know how many times a car has been rented in the past in order to perform a car check.

In addition to an object model, which takes care of the dynamics of the system, MERODE uses a kind of Entity-Relationship (ER) model to express the static aspects of the system. Both models are fully integrated: each object in the object model corresponds to one and only one entity in the ER-model. One of the cornerstones of MERODE is existence dependency, with following definition:

Object type A is existence dependent from object type B, if and only if each object of type A, during its entire life-cycle, engages in a mandatory relationship with one, only one and always the same object of type B.

As a matter of fact, all relationships in the ER-model should denote existence dependency. In our car rental company, ‘car’ and ‘customer’ can therefore not be linked directly: a car can exist without being rented by a customer and a person does not only exist by means of his renting a car. Thus, the need of existence dependency should involve the creation of a new object ‘rental’, existence depended both of ‘car’ and ‘customer’. Why this is so important is made clear by Snoeck (Sneek et al. 1996). Next to simplifying consistency checking in the model, existence dependency is a valuable alternative for the sometimes-vague concept of the Part-Of relation.

Finally, special attention is given to specification of business constraints. A distinction is made between two different types of constraints. Sequence constraints (e.g. a car cannot be rented before it is bought) are expressed by sequence diagrams (as in the Jackson System Development method JSD) or by finite state machine charts. Data constraints (e.g. a client can only rent 3 cars at the same time) are expressed in a declarative way. By modelling constraints, the object model itself will be able to maintain integrity, as advocated for building models by Galle (1995). The different aspects are described in the most appropriate way, keeping in mind that consistency between the different notation techniques is essential. Thus, an object model handles the dynamics of the system, an Existence Dependency graph scheme - the mentioned ER-like scheme - takes care of the static relationships between objects, and finally sequence and data constraints are described by JSD diagrams and Eiffel-like notations. All this leads to the final ADT (Abstract Data Type) description of the objects.

The MERODE methodology was considered a firm base in order to start the research on the construction of the core object for architectural design in the early
design phase. Due to the strict distinction between business modelling and the rest of the model, user input (by using a CAD package), output (architectural drawings, data for computational tests, etc.) and creation of information objects (essential for version control) could come in to play in a later research phase. Different views on the model by different members of a collaborative design group do not influence this business model and can be specified later.

2.2 Development of the core object model

The specifications of the object model were deduced from the theoretical framework. Parallel with this framework, a state-of-the-art in product modelling and integrated efforts was made up. Finally, we called upon the design expertise at our department. The following paragraphs present the premises of the object model. They discuss aspects often ignored in current research initiatives. More information on the research project as a whole can be found in Hendrix (Hendrix et al. 1998).

2.2.1 In pursuit of ‘good architecture’

The object model should be able to describe architectural designs in a full-fledged way. We do not want to confine ourselves to evident, routine design and ‘poor’ architecture, but we are aiming at a digital design environment that can lead to a creative and qualitative architecture. The designer cannot disclaim responsibility in this matter, but should at least not be limited by the CAAD environment. In this respect, seminal examples of Belgian architecture - as open to question the definition of good architecture may be - validated our model. Although we know that the realisation of this ambition is closely connected with flexible and user-friendly modelling tools, we limited ourselves in this phase to the - as complete as possible - description of a full-fledged design. Important aspects are:

- the description of boundaries between design elements: between two physical objects, between a physical object and a space, between two spaces
- considering architectural design as a process: one and the same design object can be represented in different ways during this process
- a general object definition capable of describing both physical design objects and abstract spaces
- avoiding restrictions in the use of geometrical shapes to depict building objects

These basic considerations have an important impact, even on the static description of the contemplated object model. As a consequence they were validated in an early stage of the project. It led to the development of a first prototype by means of a relational database, providing easy input and checking of relationships between objects. Images of this first prototype are shown in figures 2 and 3.
In this example, the general applicability of the model is tested by the design for a single-family row house (architect Mauro Poponcini, Antwerp 1994). The core object structure is implemented in Microsoft Access; AUTOCAD R14 takes care of
the graphical representation. The connection between the central object description and the on-screen representation is made by adding an attribute to the geometrical drawing objects holding the AutoCAD entity's Handle or Entity Name.

Also the principle of multiple representations has been tested and its validity was proven. In this product model one and the same staircase for example is depicted by a sloping plane in the early phase and by a solid visualising each step at a later design phase. Both representations are kept in the database. The kind of representation is dependent both of the current design phase and the scale of zooming. This aspect will be further explained in paragraph 2.2.4.

This first prototype has proved two things. On the one hand, the nature of the implementation (relational or object oriented approach) is not dictated by the conceptual model. On the other hand, the model is capable to describe real projects and is not limited to trivial designs.

2.2.2 Supporting the early design phases

The importance of supporting the early design phases has been mentioned earlier in this text and has following consequences:

First, the design environment should permit the description of a design from scratch by offering 'primitive' architectural objects as well as more elaborated ones. Primitive objects can be further specified in the course of the design process. How the model covers this aspect will be shown in paragraph 2.2.4.

Second, the environment should offer appropriate tests for the early phases to turn traditional post-design computation into 'testing-while-designing'. At the moment, our research group is developing a daylight and insolation test for the early design phases. It uses the core object model to extract the appropriate data and leads to both a computational and graphical evaluation of the design at hand. Within the scope of this paper, we will not pursue this matter further.

2.2.3 Different design approaches - different design entries

The design environment is not limited to a given building programme (school, hospital, town planning, office building...) or a given set of computational tests (light, insolation, statics, cost...), neither does it want to impose a certain way or methodology of design upon its user. Both specifications call for the possibility to approach the design process from different point of views. We explicitly model both physical objects (from terrain, wall and floor to door-handle and chalk layer) and the abstract notion of 'space', an approach also mentioned by others (e.g. Eastman et al. 1995). On top of this duality, the designer has the possibility to handle mere geometrical forms, a possibility also defended by Ekholm (1966). This leads us to the notion of 'possible design entries'. This doesn't rule out the mutual connection between the mentioned objects, but comes down to not assigning a certain order of importance upon them. To say it the MERODE way: none of the basic elements is existence dependent of another one. These different views on the design process and the corresponding design entries were considered very important. To see how the model copes with it, we first discuss the general structure of the object class named ELEMENT.

To cover all possible aspects of a building object, we create an overall neutral object class ELEMENT (or THING) that can contain the different kinds of data
belonging to an object: graphical information, specific architectural information, additional alphanumerical information (figure 4). The last object class can for example handle the data connected with a given activity or function and thus allows the modelling of user activities.

![Diagram](image.png)

*Figure 4. Main construction of a building element*

As said before, the different components of an ELEMENT have no predefined order of importance. This makes the contemplated different design entries possible, as depicted in figure 5:

- juggling around graphical drawing objects in space without assigning an architectural meaning to them from the start (upper left arrow)
- positioning physical building elements (lower left arrow)
- positioning and modelling spaces, with a possible 'materialisation' afterwards (lower right arrow)
- organising the layout of user activities, connecting them to specific spaces afterwards (upper right arrow)
2.2.4 Architectural design as a process

Design is a process. Design objects are not defined and fixed once and for all, but evolve during the design process. Closely connected are the notions of design phases and different architectural representations. This results in an additional challenge: providing the possibility of change.

Evolution of objects has to be handled within the core object model, thus making reuse of data possible: when changing/elaborating an existing design element, as much data as possible is reused to build the new version of it, thus avoiding the necessity to reconstruct an element from scratch every time a major change occurs.

Looking at this aspect from the object-oriented point of view, designed objects appear to switch classes. A particular wall can start as a general PLANAR ELEMENT in the early design phase, become a WALL in a later one and result in a specific kind of CAVITY WALL in a final stage.

To meet the requirement of object evolution, we use the following strategy: a building element is divided in an abstract part still named BUILDING ELEMENT and a part containing the specific 'type' information, named BUILDING ELEMENT TYPE. Considering the changing wall above, we will keep the same BUILDING ELEMENT, but its type pointer will switch from PLANAR ELEMENT to CAVITY WALL during the design process. This process is hidden to the user: the object simply seems to change classes. Figure 6 clarifies this structure.

Object classes are represented by rectangles holding the class name. The small white squares suggest contract objects: a construction within MERODE to ensure the existence independence of the object classes it connects. Within the scope of this paper, it is unnecessary to enlarge on this.
The notion of changes in representation is closely connected with this object evolution (see also paragraph 2.2.1). The same figure 6 illustrates our solution.

Several architectural representations are assigned to one and the same BUILDING ELEMENT TYPE. The transition between two design phases (implying a new value for current_representation) leads to the selection of the appropriate representation: the correct member function (architectural_representation_#) is activated and the corresponding GRAPHICAL ELEMENTs will depict the new situation. Doing so, our above-mentioned ‘wall’ will be displayed as a simple line or surface, as a cuboid in space or as a very detailed representation of a cavity wall. The red arrows in figure 6 show this cycle of changing representations. At this stage, the member function architectural_representation_# uses a parameter description or sequence of drawing commands to obtain the wanted graphical output. A more convenient way to describe the often complex predefined and user-defined representations will be the topic of further research (e.g. description in a suitable script).

The example below shows a possible implementation of the core object model in the C++ programming language. AutoCAD handles input and output. Here, the aspect of different architectural representations is connected with the notion of ‘intelligent zooming’. As mentioned in paragraph 2.1, the final conceptual object description in MERODE is the ADT (Abstract Data Type). To illustrate this, we provide the ADT of the core object class BUILDING ELEMENT. Next, the C++ implementation of the same object class as used in the above prototype is shown.
**Figure 7.** The multiple representation principle (implementation by Bart Anrijs)

**Figure 8.** Sample code of an object ADT by MERODE and its C++ implementation
2.2.5 Clear separation between the core model and its implementation

An important starting-point is the neutral description of the core object model. This neutrality has a double meaning:

- The fact that the core object model is developed with no particular design domain or design test in mind
- The fact that the core description of objects is disconnected from its input and output, notably its on-screen realisation or implementation (and thus from the graphical packages that take care of an object’s graphical representation)

This paragraph discusses the second kind of neutrality.

To obtain the possibility to work with different graphical applications (e.g. using AutoCAD graphical entities or Open Inventor API), we apply the typical 'bridge' principle (Gamma et al., 1995): a neutral GRAPHICAL ELEMENT is separated from its application dependent implementation by building an intermediate GRAPHICAL REALISATION (see Figure 9).

![Diagram of graphical element and realization](image)

Figure 9. Separation between a graphical element and its on-screen representation

Switching the graphical package to provide input and output implies that the correct GRAPHICAL REALISATION object has to be called. To use a new graphical package you only need to create the appropriate GRAPHICAL REALISATIONs: the core model itself remains intact and does not require changes.

This principle was tested by linking the graphical elements of the core object model to two different graphical output packages: on the one hand AutoCAD by means of ObjectARX (see also the example in 2.2.4), on the other hand Open Inventor. An illustration of a prototype using Open Inventor and steered by a central C++ implementation is shown in figure 10.
2.3 Overall picture of the developed core object model

We like to conclude this section with an overall picture of the developed core object model. In the (simplified) scheme of figure 11 the mentioned principles can be distinguished. Yet, this existence dependency graph is only one of the different facets of the complete MERODE model: here only the core objects and the static relations between them are shown. As mentioned before, contract objects are depicted by a little white square. Only the main scheme is given, the grey object classes stand for an embedded hierarchy of object classes. The system as a whole will be thoroughly covered in a forthcoming PhD thesis on the subject.

The scheme is presented in the MERODE notation method. We considered the use of the more general Express-G notation, but decided to stick with the MERODE method in its entirety. With respect to content, this does not make a difference.
3. **CONCLUSION**

This paper discusses the core object model for the architectural design process, developed within the framework of the IDEA+ project (Integrated Design Environment for Architecture) of the CADLAB at the KULeuven. The model was developed with the aid of MERODE, an object-oriented analysis method. After arguing the need of such a model, its strong points were revealed. These points were illustrated by means of examples taken from prototypes developed at the CADLAB. Further research topics are the parameter/script description and representation of building types and the user-friendly interface between the core model and the designer. Although a lot of work still lies ahead of us, the core object model has proven to be a strong base for further developments.

4. **REFERENCES**


Eastman, C., Jeng, T.S., Chowdbury, R., 1997, "Integration of design applications with building models", CAAD futures 1997, Proceedings of the 7th International Conference on...
Gamma, E., Helm, R., 1995, "Design Patterns, Elements of Reusable Object-Oriented Software" (Addison Wesley)
Neuckermans, H., 1992, "A conceptual model for CAAD", Automation in Construction, 1 (1) 1-6 (Elsevier)
Verhelst, M., 1996, "Object-oriented application development with M.E.R.O.DE.", course text, Department of Applied Economics, KU Leuven